Pràctica 2: Recollida de dades

Aquest treball ha sigut realitzat a partir del joc que vam desenvolupar a projecte dos. El joc desenvolupat va ser Age of Empries II, un joc de estratègia a temps real ambientat en l ‘era medieval en el que el jugador lluita contra la intel·ligència artificial per derrotar les seves unitats.

Al no tenir part online on existeixin varis usuaris, les dades analitzades son les que generen les diferents unitats que interactuen durant la partida mitjançant el combat. El jugador pot generar unitats mitjançant recursos i temps i seguidament son enviades a combatre. Les dades que es guarden serveixen tant per analitzar la producció d’unitats com el seu rendiment en combat.

Obtenció de dades:

Variables analitzades:

* EntityID: ID de la entitat que s’especifica quan es generada i s’utilitza per contar el total d’unitats generades al llarg de la partida.
* EntityType: Tipus de la unitat especificat per l’usuari.
* Name: Nom de la unitat generat aleatòriament al generar la unitat.
* Resources: Cost en recursos de la unitat.
* KillScore: Puntuació que ha aconseguit la unitat. Càlcul realitzat a partir del mal i les morts realitzades.
* TrainTime: Temps que l’usuari ha necessitat per crear la unitat.
* Kills: Morts que ha realitzat la unitat.
* DamageDone: Mal que ha realitzat la unitat.
* DamageGet: Mal que ha rebut la unitat.
* BuildingKills: Edificis que ha destruït la unitat.
* BuildingDamageDone: Mal a edificis que ha realitzat la unitat.

Per guardar aquestes variables inicialment s’ha creat una taula a MySQL amb el camps necessaris. Seguidament s’ha adaptat el codi del projecte per realitzar una crida a MySQL cada cop que una unitat del jugador es morta.

A continuació es mostra els segments de codi mes representatiu en quan a l’enviament de dades a MySQL.

Codi implementat:

void EntityManager::KillUnit(Unit\* \_unit)

{

if(\_unit->ally == true)

{

App->server->mysqlDatabaseGateway.SaveUnitData(\_unit);

}

Dead\_list.Add(\_unit);

}

void MySqlDatabaseGateway::SaveUnitData (const Unit & \_unit)

{

DBConnection db(bufMySqlHost, bufMySqlPort, bufMySqlDatabase, bufMySqlUsername, bufMySqlPassword);

if (!db.isConnected())return;

DBResultSet res;

char ID\_buffer[64];

\_itoa\_s(\_unit.ID, ID\_buffer, 10);

char type\_buffer[64];

\_itoa\_s(\_unit.type, type\_buffer, 10);

char resources\_buffer[64];

\_itoa\_s(\_unit.resources, resources \_buffer, 10);

char killscore\_buffer[64];

\_itoa\_s(\_unit.killscore, killscore \_buffer, 10);

char traint\_buffer[64];

\_itoa\_s(\_unit.traint, traint\_buffer, 10);

char kills\_buffer[64];

\_itoa\_s(\_unit.kills, kills\_buffer, 10);

char damage\_done\_buffer[64];

\_itoa\_s(\_unit.damage\_done, damage\_done\_buffer, 10);

char damage\_get\_buffer[64];

\_itoa\_s(\_unit.damage\_get, damage\_get\_buffer, 10);

char build\_kills\_buffer[64];

\_itoa\_s(\_unit.build\_kills, build\_kills\_buffer, 10);

char build\_damage\_buffer[64];

\_itoa\_s(\_unit.build\_damage, build\_damage\_buffer, 10);

std::string sqlStatement;

sqlStatement =

"INSERT INTO AoE (EntityID,EntityType,Name,Resources,KillScore,TrainTime,Kills,DamageDone,DamageGet, BuildingKills,BuildingDamageDone)

VALUES ('" + ID\_buffer + "', '" + type\_buffer + "', '" + \_unit.name + "', '" + resources\_buffer + "', '" + killscore\_buffer + "', '" + traint\_buffer + "', '" + kills\_buffer + "', '" + damage\_done\_buffer + "', '" + damage\_get\_buffer + "', '" + build\_kills\_buffer + "', '" + build\_damage\_buffer + "');";

// insert unit data

db.sql(sqlStatement.c\_str());

}

Resultat obtingut:

Segment de 26 mostres d’un total de 533.

